INTERNATIONAL JOURNAL OF MATHEMATICS AND COMPUTERS IN SIMULATION
DOI: 10.46300/9102.2020.14.6 Volume 14, 2020

Mobile Robot Simulation and Navigation in ROS and Gazebo

Denis Chikurtev
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Sofia, Bulgaria
dchikurtev@gmail.com

Received: April 30, 2020. Revised: September 17, 2020. Accepted: September 28, 2020. Published: September 30, 2020.

***Abstract*— mobile robots are entering our daily lives as well as in the industry. Their task is usually associated with carrying out transportation. This leads to the need to perform autonomous movement of mobile robots. On the other hand, modern practice is that the planning of most processes is done through simulations. Thus, various future production problems can be anticipated and remedied or improved. The article describes the creation of a mobile robot model in the Gazebo simulation environment. Specific settings and features for running a mobile robot in autonomous navigation mode under the robot operating system are presented. The steps for creating a map, localization and navigation are presented. Experiments have been conducted to optimize and tune the parameters of both the robot model itself and the simulation control parameters.**

Keywords*— mobile robot, simulation model, autonomous navigation, ROS, Gazebo

I. Introduction

Service robotics is a very popular area that has undergone significant development in recent years. Service robots are widely used. They are becoming more and more commonplace in our daily lives as well as in various fields of industry, healthcare, medicine, education, construction, entertainment and more [1, 2, 3]. Most service robots are mobile because they can perform their tasks related to assisting humans and / or machines [4, 6]. The mobility of the robots enables them to perform the tasks for which they are intended without difficulty [2].

Nowadays there is a very wide variety of mobile robots. The main types of mobile robots, according to their locomotion, are divided into wheeled, chain, walking and floating. The subject of research in the article are wheeled mobile robots. They are among the most common and are relatively easy to operate, unlike walking robots. Wheeled mobile robots are equipped with different types of wheels - standard, omni wheels, mecanum wheels and more. There are different configurations of mobile platforms according to the location and number of wheels. Thus, we have differentially positioned wheels (such as tank-type platform) with one or two caster wheels, triangular omni-wheel platforms, standard four-wheel platforms with mecanum wheels and other specific and unique types.

All these varieties and configurations of mobile platforms are characterized by specific kinematic and dynamic characteristics. These specifics are at the heart of managing a mobile robot. They determine the behavior of the platform and controllers. In a simulation model, these characteristics must be correctly set in order to get as close as possible to the real model.

Simulating a mobile robot can be of great benefit. On the one hand, having a robot model in a simulation environment can be used by many people without the need for everyone to own a physical robot. On the other hand, a number of experiments and studies can be conducted in this way without compromising the security of humans, the robot or the objects that the robot handles and interacts with. Last but not least, running time through simulation can be greatly reduced, since many activities such as preparing for experiments or restoring the robot's initial parameters and the characteristics of the environment in which it operates are eliminated.

Another major problem with mobile robots is their autonomous movement. Autonomous navigation involves localization, path planning, and motion control of the mobile platform [8]. All these problems have a number of solutions and can be investigated in a simulation environment. The article describes the creation of a simulation model of a differential drive mobile platform equipment with a laser sensor and a simulation controller in a Gazebo environment. Then describes the use of the navigation package ROS and the necessary tools and programs for its implementation.

II. ROS and Gazebo

The robotics operating system is widespread and used by all those involved in robotics. The system allows the use of numerous ready-made packages, tools, models and libraries for robots [6]. And because it is open source each user has the ability to use and modify the right one for him pack, tool or library.

The Gazebo simulator can connect directly to the ROS through special packages. These packages provide the necessary interfaces to simulate a Gazebo robot using ROS messages, services, and dynamic reconfigure.

In order to work correctly with a robot, PAC needs a description of the kinematics of the robot. In this way, trajectories, navigation, and more can be planned and executed. ROS way of describing a robot is by specifying its properties in URDF (Universal Robot Description Format) files. URDF supports XML and *xacro* (XML macro) languages. Xacro code is easier to implement, maintain and has better readability. To use a URDF file in Gazebo, some additional simulation-specific tags must be added to work properly with Gazebo.

The relationship between ROS and Gazebo is the same as that between ROS and the hardware of a real robot. The

controller in ROS receives data on one topic from both models and publishes data on another topic to both models. Simulation and real robot control can be performed at the same time. This way, their behavior and work can be easily compared. In fact, ROS makes no difference whether it controls a real robot or a simulation model of a robot. The important thing here is to implement the correct nodes and to have the necessary topics for communication.

Figure 1 shows the structure of the Gazebo simulation model. It consists of a model with a description of the robot, its plugins and Gazebo libraries. The model receives data from the Joint Command Interface, and as a feedback sends data through the Joint State Interface. The same applies to the hardware model (Figure 2), which is connected to the ROS controller via the same interfaces. The difference between the two models is that the hardware model includes components such as an embedded controller, actuators and sensors, which in Gazebo are replaced by simulated ones by plugins and libraries. The ROS controller is shown in Figure 3. It processes the data received from any model and sends control commands accordingly [13].

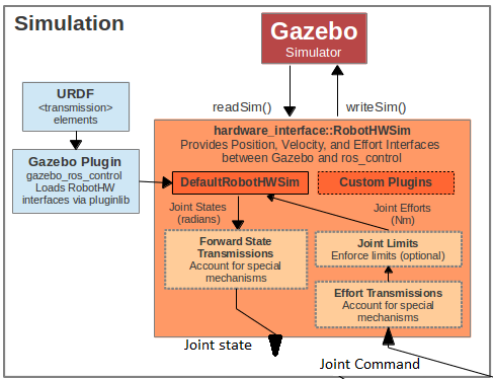

Fig. 1. Principal Robot Simulation in Gazebo.

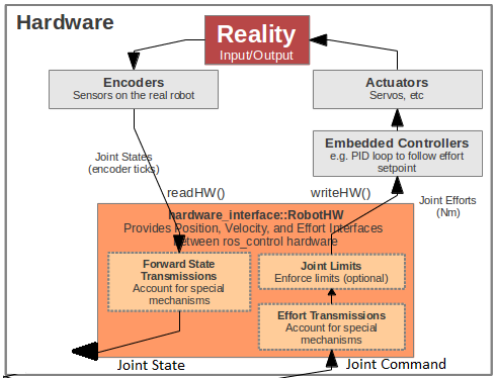

Fig. 2. Principal Robot Hardware.

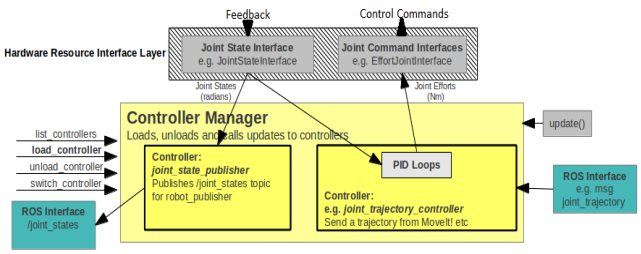

Fig. 3. ROS Contrrller Manager.

III. Design of a Mobile Robot in Gazebo

Robot simulation is at the heart of creating and testing robot models [5]. A well-designed simulation model enables testing algorithms, controllers, robot design, artificial intelligence training, and more. The Gazebo platform offers the ability to accurately and effectively simulate robots in complex environments, both outdoors and indoors. Gazebo offers physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces for both users and programs.

A few key features of Gazebo include:

- multiple physics engines,
- a rich library of robot models and environments,
- a wide variety of sensors,
- convenient programmatic and graphical interfaces

Creating a robot simulation model can happen by using Gazebo's graphical user interface or by directly programming the robot's parameters and properties in files. In both cases, the features and rules for creating a robot and describing the parameters of its units are the same. There are a number of steps that can be taken to create a mobile platform. The article describes the minimum number of steps required to create a working model of a mobile robot.

- creation on the basis of the platform: the following properties are defined - shape, dimensions, mass, moment of inertia (according to the form: cylinder, cube, parallelepiped), type of unit: parent unit;
- creation of the driving wheels: according to the type of platform, the required number of wheels are created with the following properties - shape, dimensions, mass, moment of inertia, type of unit: child unit;
- creation of caster wheels: they are set in the shape of a ball with defined radius and positioning;
- set joints between the base and the driven wheels: set the type of joint, set the type of units - the base is the parent, the rest are children, set the axis of rotation of the wheel, positioning the child units relative to the base;
- fixing joints between the base and the caster wheels: because they rotate in all directions, we choose the type of the joint to be ball, then position them against the base;
- adding sensors: Sensors are described like other units and the most important thing for them is to determine the connection to the joint to the base. Then a description of the sensor properties is added such as: update rate, resolution, range, scan angle;
- adding plugins: in our case we add a gazebo plugins to control the differential platform and scanning the area with the lidar. Gazebo controllers set a number of parameters such as update rate, connection names, wheel diameter, wheel spacing, torque, range, scanning angle, input and output topics.

The dimensions and parameters of all robot components are presented in Table 1. There are two files in which the robot model information is programmed and stored. One file is of

type xacro and contains information about the size and location of all links and joints of the robot. Each individual element of the robot is represented as a link with a specific name. The link is described with the following properties: collision, visual and inertial. After all the links have been described, the joints should be described in order to obtain the connection between the links.

TABLE I. ROBOT'S PARTS DIMENTIONS.

Element	Shape, Dimensions	Parameters
base	cylinder, radius = 0.2 m, length = 0.05 m	mass = 10 kg
driving wheels	cylinder, radius = 0.08 m, length = 0.03 m	mass = 2 kg
caster wheels	sphere, radius = 0.04 m	mass = 0.5 kg

The other file is a gazebo extension, it contains information about how each element looks in the simulation, the plugins used and their settings, as well as the properties and settings of all the sensors. The differential_drive_controller plugin is used to drive the mobile robot. The controller parameters are:

<legacyMode>false; <alwaysOn>true; <updateRate>30; <leftJoint>left_wheel_hinge; <rightJoint>right_wheel_hinge; <wheelSeparation>0.4; <wheelDiameter>0.16; <torque>15; <commandTopic>cmd_vel; <odometryTopic>odom; <odometryFrame>odom; <robotBaseFrame>chassis.

Wheel and platform color references have been added. The following is a reference for the laser scanner and its features:

<pose>0 0 0.06 0 0 0; <visualize>false; <update_rate>40; <scan> <horizontal><samples>720; <resolution>1; <min_angle>-3.14159265; <max_angle>3.14159265; <range><min>0.2; <max>25.0; <resolution>0.01; <noise><type>gaussian; <mean>0.0; <stddev>0.01.

Finally, we use the gazebo_ros_head_rplidar_controller plugin to publish the scanned data into a topic: <topicName> / mybot / laser / scan; <frameName> laser.

There is another way to describe the elements is to import a mesh file with dae extension. Such a file can be exported from a drawing in Blender. This type of file describes the characteristics of the element in great detail. In our case, the description of the RPLidar A2 laser sensor is inserted [11].

After completing all these steps, the mobile platform is ready for use. Figure 4 shows the platform designed in Gazebo. After launching the robot model in ROS and Gazebo, we can check what the active nodes and topics are. So far, we have two output topics - odom of odometry, LaserScan of lidar, and one input topic (cmd_vel) for wheel control.

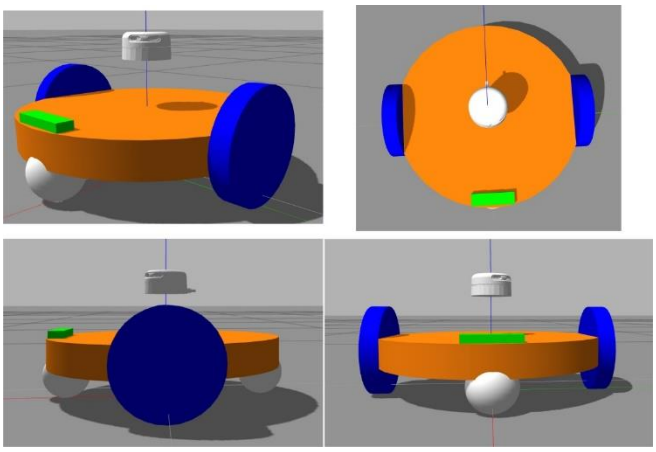

Fig. 4. 3D Simulated Model of Differential Drive Mobile Robot.

The final stage of modeling in Gazebo is to create a model of a simulated world. Thus, the robot will be located in a certain environment with its properties. To make it easier for developers, Gazebo offers a significant range of ready-to-use items and even buildings. Therefore, we can easily create our world and save it. The created world information is saved in a .world file. Finally, in order to launch the robot in the world we create, we must start together the robot model and the world file in a .launch file (Figure 5).

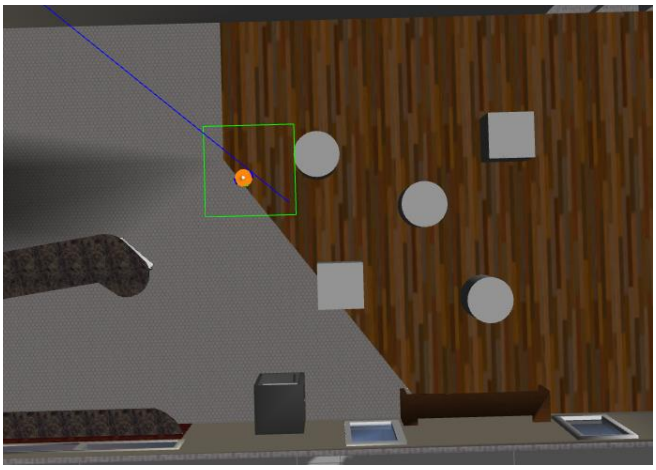

Fig. 5. Robot Model in Simulated World.

IV. ROS NAVIGATION STACK

Once we have made a simulation model of the robot, we can configure it to operate at autonomous navigation. Several new steps should be taken to achieve this. These steps are determined by the navigation package requirements.

The principle of operation of the navigation package is presented in Figure 6. Autonomous navigation is performed at the move_base node [12]. This node receives the desired destination data from the moce_base_simple / goal topic. Then move_base reads the data from the odom, LaserScan, tf - transform library [7], and GetMap topics, processes the data and plans the path. Finally, the necessary commands are published in the cmd_vel topic for controlling the mobile platform.

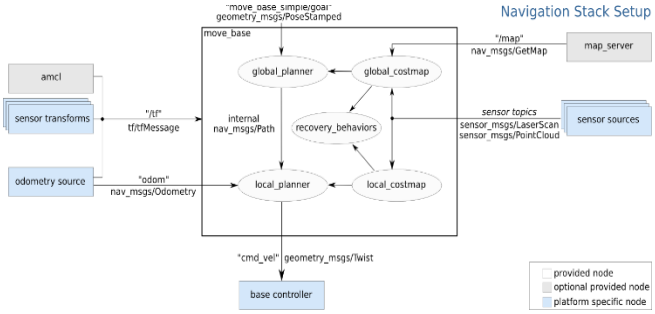

Fig. 6. ROS Navigation Stack Configuration.

In order for navigation to work correctly, both the mapping and localization packages must be configured [8]. These two packages are part of the packages in the ROS and can be used ready-made. The Mapping package is used to create and retrieve map information [9]. The AMCL package provides self-localization algorithms [10]. In addition, some global and local planer parameters, as well as global_costmap and local_costmap parameters can be set. When the entire navigation system is already configured, we move on to experiments.

Initially, a map of the work environment should be created. The map creation process is done manually. The following programs and nodes are started:

- launch the model of the robot in the simulated world;
- slam_gamapping node that reads odometry and laser scanner data and returns data to create a new map;
- tf node [7];
- node for manual control of the robot, via keyboard or joystick;
- rviz interface for rendering the map made;

after the map is ready, we save it. Figure 7 shows the map made.

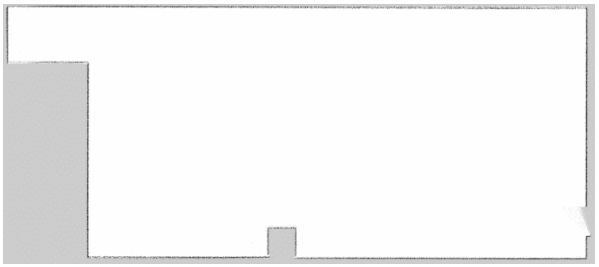

Fig. 7. Created Map of the Gazebo World.

When the map is ready, we can start the autonomous navigation mode. To do this, the following programs should be started:

- launch the model of the robot in the simulated world;
- the nodes in figure 6: map_server, tf, amcl, move_base;
- rviz interface to visualize and set desired destinations.

So now we have a working simulation model of the robot under the control of PAC, in the mode of autonomous navigation. To make sure we have all the nodes we need, we can open rqt_graph. An interface opens showing all running nodes and topics (Fig. 8). This check is very useful because we can easily find out if the connections between the nodes are correct and if there are missing links.

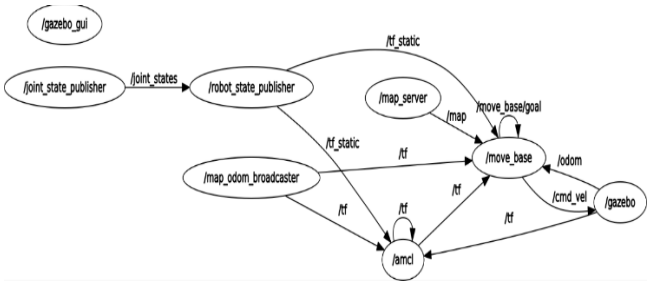

Fig. 8. Active ROS Nodes in Navigation Mode.

V. Experiments and Results

As only the minimum requirements for operating a robot in autonomous navigation mode are met, the robot is not expected to move perfectly and always achieving the set coordinates. However, experiments can be performed to verify that the robot can perform the tasks.

The experiments performed are of two types. The first type of experiments checks whether the robot can move independently in a room without obstacles. In the second type of experiments, additional objects were added into the room and then we check if the robot could detect obstacles and avoid them.

Both types of experiments were successful. As expected, when navigating the robot in an empty room with no obstacles added, all experiments are successful. The system successfully locates the robot, generates a path, and navigates the robot. The robot reaches the set position each time. The navigation system has no difficulty since it has a lot of free space.

However, this changes when there are additional obstacles. Figure 5 shows the added cubes and cylinders in the middle of the room. It should be noted that they were added after the card was already created and do not exist on the card itself, as can be seen in Figure 5. In these experiments, the navigation system itself recognizes the obstacles and even during the movement of the robot changes the planned path if necessary.

Figures 9, 10, 11, 12 show the steps of completing one of the tasks. The starting position of the robot is the beginning of the blue line. The destination given is the position that the robot has reached in Figure 12. The robot is shown in orange, in blue is the planned path by global_planner, in the green is planned local path by local_planner, in red and purple are the contours of the map and obstacles.

In black, the zone of collision is indicated as a shadow, i.e. the maximum distance that the robot can approach to an outline / obstacle. The distance to the contours can be adjusted depending on the settings and behavior of the robot.

Just before the experiment started, we added a new obstacle - a cylinder in the middle among others. It can be seen that the laser scanner recognizes the contour of the cylinder, but the navigation system has not yet added a black shadow because the object is not yet within the scope of local_planner. However, when the robot approaches and the object become within range, it is recognized as a new object and the system corrects the original path.

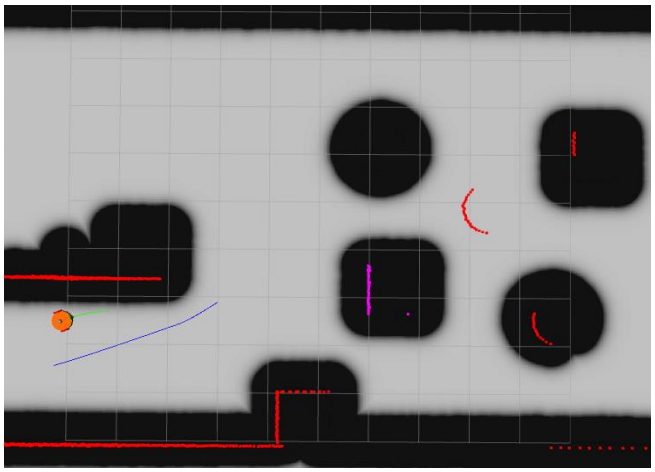

Fig. 9. Navigation experiments – starting.

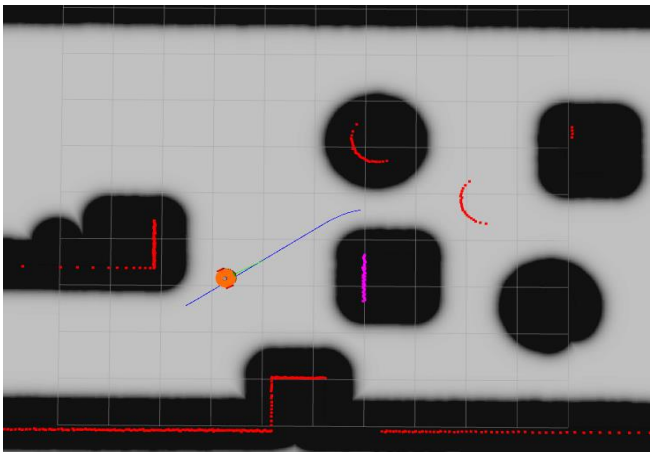

Fig. 10. Navigation Experiments - the robot aligns with the planned path.

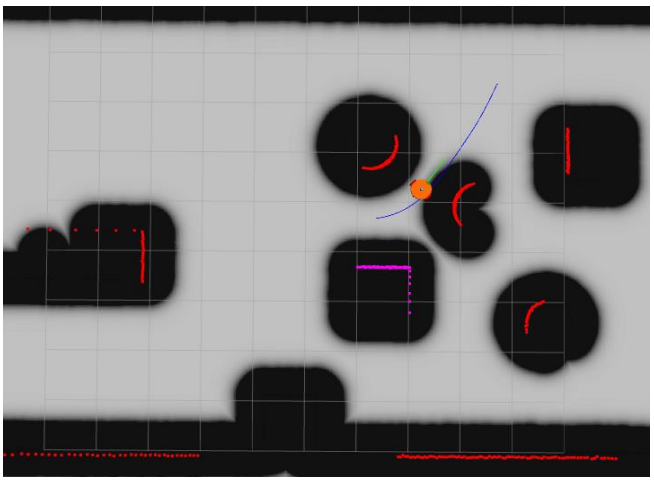

Fig. 11. Navigation Experiments – recognition and avoiding the new obstacle.

In some of the experiments, the robot failed to reach its target. In these cases, the robot falls into the area of collision with an object after deviating too much from its task or after not recognizing an obstacle in time. This happens for two reasons: lack of a controller for robust robot control and delay in refreshing rate of the local_costmap.

While for the first reason an ROS controller for speed control of the robot must be set up, the second is easily corrected by changing the refresh rate parameter in local_costmap.

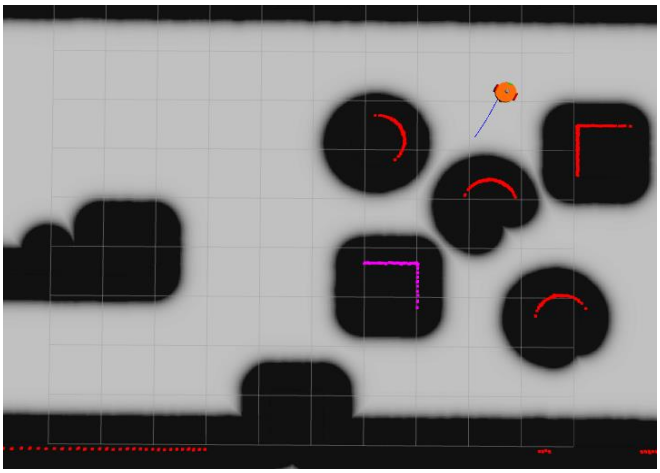

Fig. 12. Navigation Experiments – reach the final goal.

In addition, it is advisable to adjust the base_local_planer parameters because their default values may not be appropriate for each platform. The parameters in base_local_planer are maximum and minimum velocity on the x and y axes, maximum and minimum rotation speed and acceleration limits on the three axes x, y and z.

VI. Conclusion

The Robot Operating System and the Gazebo Platform offer very good capabilities for creating and controlling robots. The article describes the main steps for creating a simulation model of a mobile robot in ROS and Gazebo. The model developed is a differential robot with two caster wheels and is equipped with a laser scanner. The ROS navigation system was then configured to manage the created robot. The experiments show that without making special modifications to the settings of the mapping, localization and navigation packages, the robot can move independently in the simulated environment.

However, to achieve accurate path tracking, smooth movement and precise positioning, it is necessary to add additional systems such as a speed controller and an inertial sensor. This is planned to be as future work on this study.

Acknowledgment *(Heading 5)*

The research presented in this paper was supported by the Bulgarian National Science Fund under the contracts No. KP-06-M27/1 – 04.12.2018. The work was partially supported by the Bulgarian Ministry of Education and Science under the National Research Programme "Young scientists and postdoctoral students" approved by DCM # 577 / 17.08.2018.

References

[1] Zheng, Chuanqi, and Kiju Lee. "WheeLeR: Wheel-Leg Reconfigurable Mechanism with Passive Gears for Mobile Robot Applications." In 2019 International Conference on Robotics and Automation (ICRA), pp. 9292-9298. IEEE, 2019.

[2] Kim, Pileun, Jingdao Chen, Jitae Kim, and Yong K. Cho. "SLAM-driven intelligent autonomous mobile robot navigation for construction applications." In Workshop of the European Group for Intelligent Computing in Engineering, pp. 254-269. Springer, Cham, 2018.

[3] Arvin, Farshad, Jose Luis Espinosa Mendoza, Benjamin Bird, Andrew West, Simon Watson, and Barry Lennox. "Mona: an affordable mobile

robot for swarm robotic applications." In UK-RAS Conference on 'Robotics and Autonomous Systems, pp. 49-52. 2017.

[4] Ambrus, Rares, Nils Bore, John Folkesson, and Patric Jensfelt. "Autonomous meshing, texturing and recognition of object models with a mobile robot." In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5071-5078. IEEE, 2017.

[5] Takaya, Kenta, Toshinori Asai, Valeri Kroumov, and Florentin Smarandache. "Simulation environment for mobile robots testing using ROS and Gazebo." In 2016 20th International Conference on System Theory, Control and Computing (ICSTCC), pp. 96-101. IEEE, 2016.

[6] Pietrzik, S., and B. Chandrasekaran. "Setting up and Using ROS-Kinetic and Gazebo for Educational Robotic Projects and Learning." In Journal of Physics: Conference Series, vol. 1207, no. 1, p. 012019. IOP Publishing, 2019.

[7] Foote, Tully. "tf: The transform library." In 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA), pp. 1-6. IEEE, 2013.

[8] Guimarães, Rodrigo Longhi, André Schneider de Oliveira, João Alberto Fabro, Thiago Becker, and Vinícius Amilgar Brenner. "ROS navigation: Concepts and tutorial." In Robot Operating System (ROS), pp. 121-160. Springer, Cham, 2016.

[9] da Silva, Bruno MF, Rodrigo S. Xavier, and Luiz MG Gonçalves. "Mapping and Navigation for Indoor Robots under ROS: An Experimental Analysis." (2019).

[10] Talwar, Dhruv, and Seul Jung. "Particle Filter-based Localization of a Mobile Robot by Using a Single Lidar Sensor under SLAM in ROS Environment." 제어로봇시스템학회 국제학술대회 논문집 (2019): 1112-1115.

[11] Guirguis, Silvana, Mark Gergis, Catherine M. Elias, Omar M. Shehata, and Slim Abdennadher. "ROS-based Model Predictive Trajectory Tracking Control Architecture using LiDAR-Based Mapping and Hybrid A* Planning." In 2019 IEEE Intelligent Transportation Systems Conference (ITSC), pp. 2750-2756. IEEE, 2019.

[12] ROS Navigation: http://wiki.ros.org/navigation/Tutorials/RobotSetup?action=AttachFile&do=get&target=overview_tf.png

[13] Gazebo ROS Control: http://gazebosim.org/tutorials?tut=ros_control&cat=connect_ros